Reinforcement Learning-based Collision Avoidance for UAV

Đorđe Jevtić, Zoran Miljković, Milica Petrović and Aleksandar Jokić

***Abstract*—One of the significant aspects for enabling the intelligent behavior to the Unmanned Aerial Vehicles (UAVs) is by providing an algorithm for navigation through the dynamic and unseen environment. Therefore, to be autonomous, they need sensors to perceive their surroundings and utilize gathered information to decide which action to take. Having that in mind, in this paper, the authors designed the system for obstacle avoidance and also investigate the elements of the Markov decision process and their influence on each other. The flying mobile robot used within the considered problem is quadrotor type and has an integrated Lidar sensor which is utilized to detect obstacles. The sequential decision-making model based on Q-learning is trained within the MATLAB Simulink environment. The simulation results demonstrate that the UAV can navigate through the environment in most algorithm runs without colliding with surrounding obstacles.**

***Index Terms*—unmanned aerial vehicles, collision avoidance, reinforcement learning, Q-learning.**

I. Introduction

Unmanned Aerial Vehicles (UAVs) have become increasingly widely used due to their ability to operate in remote or hazardous areas, collect data, and perform various tasks autonomously. However, UAVs face a critical challenge in ensuring safe operation, primarily when operating in close proximity to other entities. Therefore, collision avoidance systems are essential to ensuring UAV safety and have been a research focus in recent years.

Reinforcement learning (RL) is a machine learning field that has shown great promise in developing systems which have to learn from experience and adapt to changing situations. This makes RL suitable framework for the implementation on intelligent mobile robots that operate in dynamic and complex environments.

In [1], the authors presented the development and evaluation of a Deep Recurrent Q-Network with Temporal Attention, which is used in a deep RL robotic controller to enable efficient obstacle avoidance for UAVs. cGAN network is used to predict a depth map, which is then utilized to determine the optimal action for the UAV. Critical information is retained over a long sequence of observations to solve the problem of partial observability successfully. In research [2] the authors focused on utilizing the Q-learning algorithm to create a method for UAVs to learn paths and avoid obstacles. In order to deal with continuous state space fitting, a neural network is employed. Additionally, the authors propose a trap-escape strategy to aid the UAV in extricating itself from problematic situations. To address the high variance and low reproducibility of collision avoidance policies obtained by utilizing RL, [3] introduces a two-stage training method for RL-based collision avoidance. Within the first stage, the policy is optimized by using a supervised training method with a loss function that encourages the agent to follow the well-known reciprocal collision avoidance strategy, while in the second stage, it is refined by using policy gradient method.

Different from other approaches, this research aims to investigate the performance of a Q-learning-based collision avoidance system for UAVs, which is a part of the larger navigation system presented in [4].

The rest of the paper is organized as follows. Section 2 provides the Bellman equation for updating Q-value at every time step and procedure for the trade-off between exploration and exploitation. Section 3 describes the proposed action space, state space, and reward function with particular reference to minimal requirements for the Lidar sensor implemented on a flying mobile robot. Section 4 explains the simulation setup, while Section 5 presents the experimental results generated within the MATLAB Simulink environment. Finally, Section 6 concludes the paper with suggestions for future research.

II. Q-learning

Q-learning is a well-known and widely used RL algorithm, which is an off-policy, model-free, temporal difference control algorithm first introduced by Watkins in 1989. One of the main characteristics of this method is the capability for

Đorđe Jevtić, Junior Research Assistant, University of Belgrade - Faculty of Mechanical Engineering, Department of Production Engineering, Laboratory for industrial robotics and artificial intelligence (ROBOTICS&AI), Kraljice Marije 16, 11120 Belgrade 35, The Republic of Serbia (drjevtic@mas.bg.ac.rs), ORCID ID (https://orcid.org/0000-0002-6917-1663).

Dr. Zoran Miljković, Full Professor, University of Belgrade - Faculty of Mechanical Engineering, Department of Production Engineering, Laboratory for industrial robotics and artificial intelligence (ROBOTICS&AI), Kraljice Marije 16, 11120 Belgrade 35, The Republic of Serbia (zmiljkovic@mas.bg.ac.rs), ORCID ID (https://orcid.org/0000-0001-9706-6134).

Dr. Milica Petrović, Associate Professor, University of Belgrade - Faculty of Mechanical Engineering, Department of Production Engineering, Laboratory for industrial robotics and artificial intelligence (ROBOTICS&AI), Kraljice Marije 16, 11120 Belgrade 35, The Republic of Serbia (mmpetrovic@mas.bg.ac.rs), ORCID ID (https://orcid.org/0000-0002-4950-6518

Aleksandar Jokić, Teaching Assistant, University of Belgrade - Faculty of Mechanical Engineering, Department of Production Engineering, Laboratory for industrial robotics and artificial intelligence (ROBOTICS&AI), Kraljice Marije 16, 11120 Belgrade 35, The Republic of Serbia (ajokic@mas.bg.ac.rs), ORCID ID (https://orcid.org/0000-0002-7417-4244).

 ISBN 978-86-7466-970-9

direct approximation of the optimal state-value function, independent of the policy being followed. For more information regarding this algorithm, the authors refer to [5].

In every time step, the state-value function is updated using the following equation:

$$Q_{t+1}(s_t, a_t) = (1-\alpha) \cdot Q_t(s_t, a_t) + \alpha \cdot [r_{t+1} + \gamma \max Q_t(s_{t+1}, a)] \quad (1)$$

where: $Q_{t+1}(s_t, a_t)$ represents state-value function at time step $t+1$; r_{t+1} denotes the reward function; s_t represents the state at time step t; a_t represents the action chosen by the intelligent agent at time step t; γ is the discount factor ($0 \leq \gamma \leq 1$); and α is the learning rate ($0 \leq \alpha \leq 1$).

Even though the intelligent agent does not have the information about the environment dynamics at the begging of the learning process, it certainly has to know which action to take at every time step, which is achieved by introducing the initial policy into the RL framework. In this paper, ε-greedy policy is utilized for action selection process, and it is defined as shown in the following equation:

$$\pi(\mathbf{s}) = \begin{cases} \text{a random action } a, \text{with prob. } \varepsilon \\ a \in \arg\max_a Q_t(s_t, a), \text{otherwise.} \end{cases} \quad (2)$$

where ε denotes the parameter which defines the ratio between exploration and exploitation ($0 \leq \varepsilon \leq 1$). One of significant challenges that especially arises in RL is trade-off between exploration and exploitation. In this paper, the ε value is updated per the following rule:

$$\begin{aligned} &\textit{if } \varepsilon_{current\_episode} > \varepsilon_{\min} \textit{ do} \\ &\quad \varepsilon_{next\_episode} = \varepsilon_{current\_episode} \cdot (1 - \varepsilon_{decay}) \\ &\textit{else} \\ &\quad \varepsilon_{next\_episode} = \varepsilon_{\min} \\ &\textit{end} \end{aligned} \quad (3)$$

where ε_{min} is the minimal value of ε which should be defined at the begging of the training process.

III. Algorithm Design

In this section, we define action space, state space, and reward function for the design of a RL simulation model utilized for the tasks of collision avoidance, navigation, and localization in the unknown environment. It is important to emphasize that the state space needs to be defined with the data not specific to a current environment so that generalization can be achieved also in a new scenario. Therefore, the environment needs to be configured in such a way as to enable UAV to visit as many states as possible. Since the task of the intelligent agent is to learn the optimal behavior, the episode is terminated when the UAV collides with an obstacle or when the maximum number of time steps is achieved.

A. Action Space

In order to define input parameters for the RL algorithm, the action space, state space, and reward function need to be defined. On the other hand, it is essential to optimize the balance of both the action and state space dimensionality to increase the learning speed of the intelligent agent and enable the adequate generalization process adaptable to other scenarios [6]. Therefore, the action vector consists of the following three actions: going forward, going diagonally left, and going diagonally right (see Fig. 1.). It is obvious, by seeing at the action space configuration, that the forward movement is favored. The defined action space reduces the maneuverability of the agent, however, a smaller action space is one of the prerequisites for adequate convergence properties of the Q-learning algorithm.

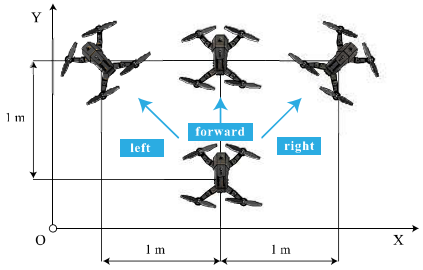


Fig. 1. Actions which an intelligent agent can select at every time step.

With the previously defined action vector, the intelligent agent needs to anticipate the sequence of actions that will lead to the state from which the currently detected obstacle is avoidable. To achieve the aforementioned, the simulation constraints that depends on action values, sensor view and range, and state space need to be defined.

B. State Space

State space is defined according to the distance data acquired by a Lidar sensor with a 360° environment view attached to the intelligent agent. It is essential to define the minimal upper bound value of the Lidar range required for a mobile robot to perform obstacle avoidance maneuver if the object is detected (see Fig. 2.). For an intelligent agent to learn the maneuver mentioned above, the considered states need to be visited a sufficient number of times during the training process in order to get as good value estimations as possible.

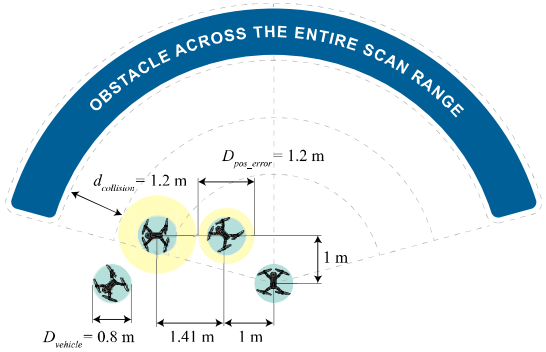


Fig. 2. Calculation of minimal upper bound value of the Lidar range.

 ISBN 978-86-7466-970-9

State space defined with all Lidar distance (increment of 2 mm) and angle values (increment of 1°) is infinitely large (3604001). Therefore, the state space needs to be discretized for a learning algorithm to converge to an optimal value. The first approximation is added to the angle of view, which is bounded to [-75,75] degrees in front of the robot (Fig. 3).

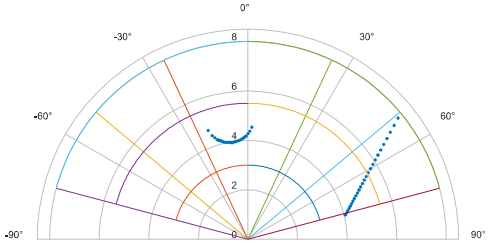


Fig. 3. Lidar measurements bounded to the [-75,75] degrees.

In the next step, state space is discretized and defined by using variables x_1, x_2, x_3, and x_4. Values of these variables depend on the area in which the obstacles are detected, defined by angle and distance from the mobile robot. Variables x_1 and x_2 are used to define the distance (d) to the closest obstacle:

$$x_i = \begin{cases} 0, & 1.2\ m \le d < 3.0\ m \\ 1, & 3.0\ m \le d < 5.5\ m \\ 2, & 5.5\ m \le d < 8.0\ m \end{cases} \quad i = 1, 2 \tag{4}$$

The main difference between these two variables is that x_1 is used to define distance in the area left to UAV's heading direction (see Fig. 4), whereas x_2 defines distance to closest obstacle in the area right to UAV's heading direction. The distance d is calculated for both sides separately.

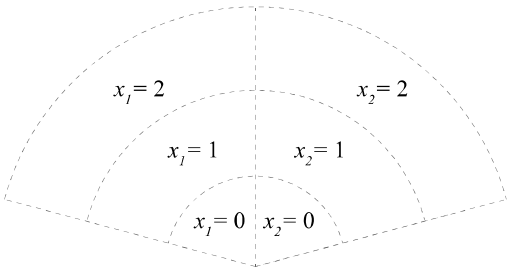


Fig. 4. State-space areas discretized by using state variables x_1 and x_2.

Variable x_3 and x_4 are defined according to the following equation:

$$x_i = \begin{cases} 0, & obstacle\ in\ h_{1(i=4)}\ /\ h_{3(i=3)} \\ 1, & obstacle\ in\ h_{2(i=4)}\ /\ h_{4(i=3)} \\ 2, & obstacle\ across\ the\ entire\ side \\ 3, & obstacle\ out\ of\ range \end{cases} \quad i = 3, 4 \tag{5}$$

where g represents the angle of the Lidar beam that detected the obstacle (see Fig. 5.), while areas h_1 : 0° ≤ g < 25° and h_2 : 25° ≤ g < 50° and h_3 : -25° < g ≤ 0° and h_4 : -50° ≤ g < -25°.

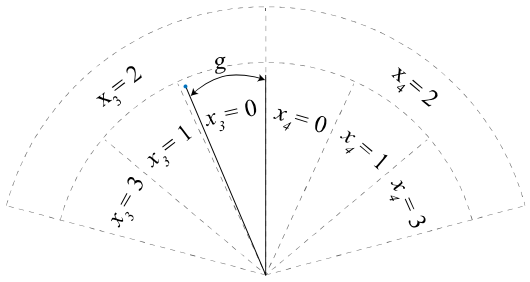


Fig. 5. State-space areas discretized by using state variables x_3 and x_4.

Finally, elements of the state-space vector are defined in the following way:

$$state_{id} = x_1 \cdot 1000 + x_2 \cdot 100 + x_3 \cdot 10 + x_4 \tag{6}$$

Since $D(x_1, x_2) = \{0,1,2\}$ and $D(x_3, x_4) = \{0,1,2,3\}$, the total number of states are 144. As mention earlier, in every time step the agent can choose one of three possible actions. Therefore, the dimensions of Q-table are 144 x 3.

C. Reward Function

To perform a training by utilizing RL algorithm, it is necessary to quantify the behavior of intelligent agent (i.e., the sequence of actions), differentiate between a set of different behaviors, and estimate the value function for every visited state-action pair. For all the previously mentioned tasks, the reward function is utilized. Reward function is also used to learn which actions are optimal in each state. In this paper, the reward function is defined as a sum of four functions (r_1, r_2, r_3 [7], and r_4), defined with equations (7) to (10):

$$r_1 = \begin{cases} +0.2, & a_t = forward \\ -0.1, & a_t = left\ /\ right \end{cases} \tag{7}$$

$$r_2 = \begin{cases} +0.2, & W_{r_2}\,\Delta d < 0 \\ -0.2, & W_{r_2}\,\Delta d \ge 0 \end{cases} \tag{8}$$

$$r_3 = \begin{cases} -0.8, & a_t = left\ /\ right \cap a_{t-1} = right\ /\ left \\ 0, & otherwise \end{cases} \tag{9}$$

$$r_4 = \begin{cases} -0.8, & \forall a_{t-n} = left\ /\ right,\ n \in \{0,1,2,3\} \\ 0, & otherwise \end{cases} \tag{10}$$

where at is the action taken at the current time step. The function r_1 is utilized to encourage the intelligent agent to move forward and discourages left or right moves. This function favors the behavior where the intelligent agent does not make unnecessary turns. The function r_2 is utilized to discourage intelligent agent from coming close to the obstacles. In (9), Δd represents the difference between the cumulative distance from the obstacles in the entire sensor view in two consecutive time steps. Whereas W_{r2} is a weighting vector that emphasizes obstacles that are closer to the center of the sensor view, i.e., that are in the intelligent agent's path. It is important to note that the positive reward in (7) must not be larger than the negative reward in (8) to

 ISBN 978-86-7466-970-9

discourage the forward movement toward incoming obstacles.

On the other hand, the positive reward in (8) needs to be larger than the negative in (7) since obstacle avoidance has priority over forward movement. The function r_3 discourages zig-zag movement, which is especially important in the domain of aerial vehicles with the limited flight time. Finally, the function r_4 is utilize to discourages movement in circles; specifically, it gives a negative reward if the agent makes four consecutive left or right actions. The entire reward function is defined with the following equation:

$$r_t = \begin{cases} -100, & \text{collision} \\ r_1 + r_2 + r_3 + r_4, & \text{no collision} \end{cases} \tag{11}$$

The reward function would be a sum of all four reward functions if the intelligent agent did not collide with the obstacle. The collision is detected if the smallest element of weighted distance to the obstacles is smaller than the threshold value which is defined as follows:

$$\min(d_w) < d_{collision}\,,\ d_w = W_t\, d_t \tag{12}$$

where W_t is a weighting vector.

IV. Simulation Setup

In this section, the employed simulation model developed within the MATLAB Simulink environment will be presented, specifically two libraries were utilized: UAV Toolbox and RL. It is important to note that RL-block can be used in event-based mode since version R2022a. Therefore this is one of the first research papers that utilizes RL-block in the mentioned mode (see Fig. 6.), but in general, also one of the first studies that use RL-block for the intelligent tasks on flying mobile robots. The prevoiously mentioned mode can be utilized when different sample time of the low-level control system and intelligent agent is required. For example, when the action is defined as the next pose that the agent should visit, the RL-block should acquire and process information only when the pose is achieved. Furthermore, this mode is also suitable when we want to investigate the flexibility of the intelligent agent's behavior, i.e. to examine the dependence between the state variables' limits defined in (4) and the values of specified actions. Accordingly, it is much easier to determine hyperparameters of the RL algorithm if we define the actions as fixed values, while the number of samples in which the action is performed/exectued is changed for every algorithm run. Another significant aspect when utilizing a reinforcement learning algorithm is the definition of an adequate sample time. Evidently, for the actions defined as in Fig. 1., this value should be as high as possible so that the agent can achieve the next pose in the shortest possible time interval. However, it should be noted that the maximum value of the sample time is conditioned by the behavior of the intelligent agent, which depends on the adopted low-level control implemented in the Simulink library. The recommended sample time for the adopted low-level control is 0.001 s, however this value turned out to be too small for training the intelligent agent. Therefore, in the next step, it was increased ten times, and after that, the accuracy in achieving the desired pose was tested, as well as the flight stability of the UAV. The agent has demonstrated the best behavior when set to move 2 m forward and slightly worse but still satisfactory when it is set to move 1 m forward, which is closer to the robot's behavior in a real environment. Therefore, it makes sense to adopt sample time which will cause reduced accuracy in achieving the desired pose. The errors should not be so large as to disturb the operation of the entire system and not small enough because it will not reflect the operation of the system in real-world. Finally, to examine the limits of the adopted low-level control, the sample time value was again increased ten times, i.e. its value was set to 0.1 s. The obtained results have showned that the flying mobile robot was not only unable to achieve the desired pose with the necessary accuracy, but also it lost stability and felt on the ground.

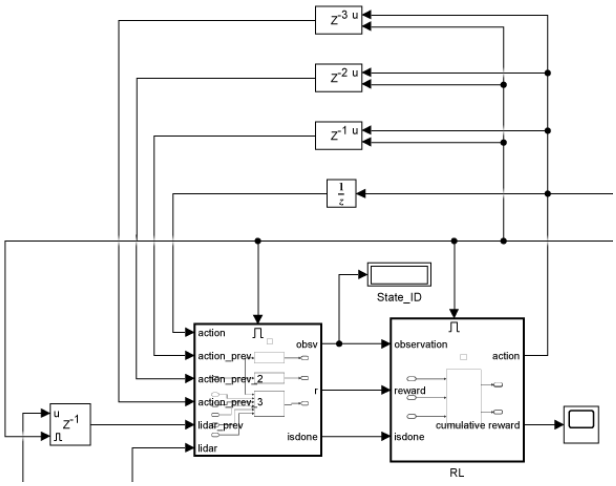


Fig. 6. Part of the complete Simulink model. As can be seen, RL agent block, observation block, reward function, and isdone function have to be the elements of enabled subsystems. Furthermore, RL agent sample time should be set to -1 in order to enable event-based simulation.

V. Experimental Results

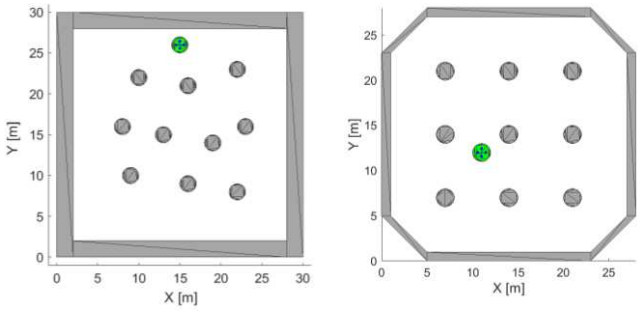
As mentioned in the previous Section, one of the prerequisites for achieving a good generalization is reflected in the number of visited states and the number of actions taken in all states of the system. The question that arises now is how can we influence the intelligent agent so that it achieves the best possible generalization. The performed simulations demonstrated that the probability of visiting a greater number of different state-action pairs is higher when the agent is set to start its movement from different initial poses rather than when the maximum number of time steps is increased. This certainly makes sense, because at the beginning of the training process, the agent mainly explores the environment by randomly choosing actions (according to the adopted ε-greedy policy). This often leads to a collision

 ISBN 978-86-7466-970-9

with obstacles and, therefore, to a reduction in the possibility of exploring some new unvisited states. Furtermore, if the agent did not visit a sufficient number of state-action pairs at the beginning of the training process, it is unlikely that it will succeed in visiting all of them in the future. Regarding solving this problem, it has a sense that actions that the intelligent agent has tried once and have led to a collision with the obstacles should be forbidden during the future exploration process, with a thorough analysis of cases in which all possible actions chosen by the intelligent agent could lead to a collision.

An additional question that arises is whether there is a connection between the configuration of the environment and the reward function. The conducted simulations showed that there is a relation between them. For example, before the implementation of the r_4 in the reward function, the agent had a tendency to drop into a local minimum during the training process, which was manifested by orbiting around the obstacle. This makes sense, since the negative reward given in expression (7) is smaller than the positive reward given in expression (8). Therefore, the maximum number of time steps was achieved while obtaining a large cumulative reward by orbiting around the obstacle, which is not desired behavior. To solve the mentioned problem, in the next step, an additional expression given in (10) was introduced. It is important to note that introducing a new element that figures within the reward function was not enough to solve the problem of achieving the local minimum because the agent again learned how to orbit around the obstacle, but this time applying different sets of actions. The conclusion was that it is impossible to define all potential cases in which this behavior occurs. However, the way to solve this problem should be sought in a different configuration of the environment itself. Certainly, it is necessary to point out one conclusion about the training processes during which this behavior was observed. Suppose the agent found itself into a local minimum in several episodes. In that case, it does not necessarily mean that it will generate identical behavior at the end of the training process due to the stochastic nature of the adopted ε-greedy policy. On the other hand, if the mentioned behavior occurred in a large number of episodes, and primarily if the agent received the large cumulative reward by orbiting around the obstacles, it is very likely that the agent will have a tendency to often fall into the local minimum after the end of the training process.

At the beginning of the algorithm testing, the environment shown in the Fig. 7.a) was adopted. The idea behind this configuration was to allow the agent to perform movement in at least a few iterations before a potential collision with obstacles occurs. The results showed that the agent learned to avoid regions crowded with obstacles and how to moving around this space in the long run, which was different from what we wanted to achieve. Thus, in the next step, the environment's configuration was changed, i.e., the distance between the boundaries of the environment and obstacles was reduced to force the agent to enter the region crowded with obstacles as soon as possible. This time, the learned behavior was more reasonable. However, it was noticed that the number of visited state-action pairs at the end of the training process was not satisfied regarding the generalization problem. In the last step, the initial pose of the agent is set to be within the region crowded with obstacles (Fig. 7. b).



a) b)

Fig. 7. Environment configuration at the beggining (a) and environment configuration at the end (b) of the algorithm testing.

Learning rate, discount factor, total number of learning episodes, the maximum number of time steps and collision distance are defined as shown in Table 1.

TABLE I
LEARNING PARAMETERS AND HYPERPARAMETERS

Max. number of episodes	600
Max. number of learning iterations (time steps)	10
Learning rate – α	0.5
Discount factor – γ	0.95
Initial epsilon value – ε	1
Epsilon decay – ε_{decay}	0.02
Min. value of epsilon – ε_{min}	0.01
Collision distance – $d_{collision}$	1.2 m

The obtained results are shown in Fig. 8. It can be seen that the learning process was successful and that convergence was achieved. However, due to the stochastic nature of the adopted ε-greedy policy training process was repeated a certain number of times in order to make an appropriate conclusion.

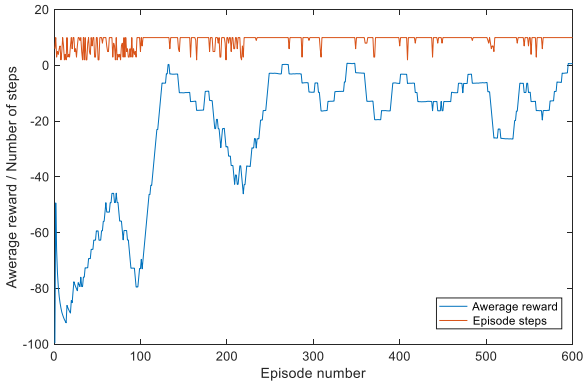


Fig. 8. Episode reward and achieved number of steps obtained by the intelligent agent during the training process.

 ISBN 978-86-7466-970-9

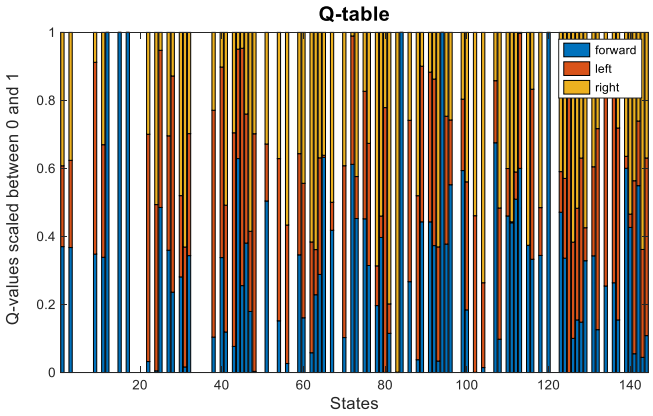


Fig. 9. Q-table after the training process.

The final output of the learning process is best illustrated by the Q-table (Fig. 9.). It can be seen that the agent has visited a large number of states during the training process and that in most of the states, the agent has learned which actions represent the optimal solution.

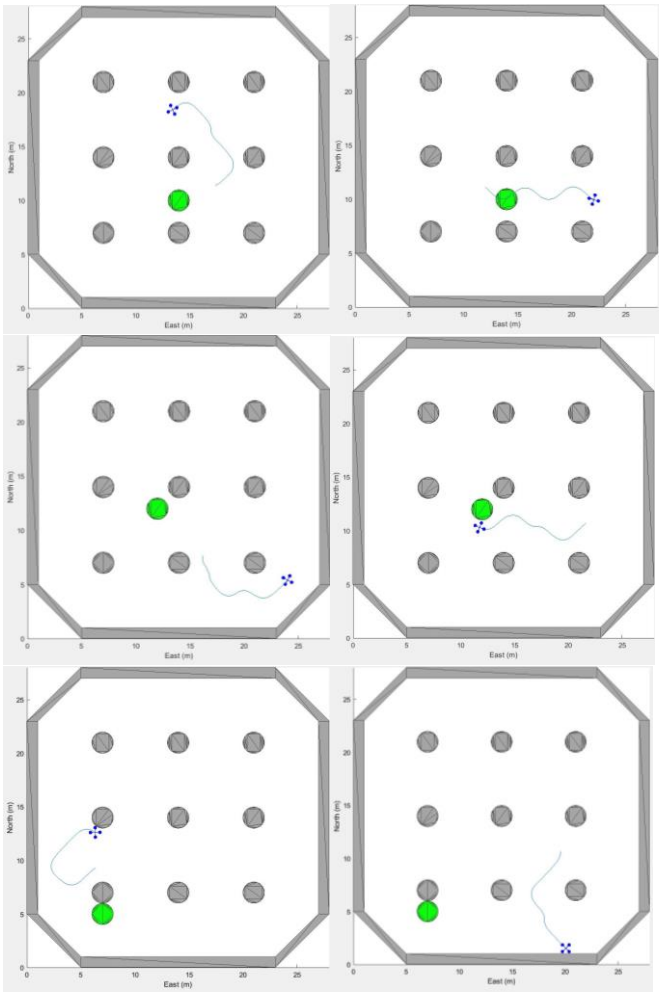


Fig. 10. The intelligent agent successfully avoids obstacles during the navigation (upper four snapshots) when starting from two different initial poses, and there are also two situations (bottom two snapshots) in which it collides with its surroundings (when starting from the third initial pose).

VI. Conclusion

In this paper, the authors presented the approach for collision avoidance of flying mobile robots based on Q-learning. Lidar data is utilized to detect obstacles in the mobile robot environment and to represent the state of the system. In each episode, mobile robot is set to a random initial pose in the static environment and trained to avoid obstacles for a fixed number of time steps. Furthermore, the process of designing an entire obstacle avoidance system is explained in detail. The obtained simulation results have shown a correlation between the state space, action space, reward function, and environment configuration as well as how defining one element influences the other. Moreover, after the learning process is completed, the generated Q-table is sufficient for the obstacle avoidance problem for majority of initial poses. In the near future research, it is planned to find the answers to the question of how the state space should be defined for arbitrarily defined actions and what the limits of such system in terms of the agent's maneuverability in the environment are. Since uncertainties and noises are more dominant in UAV scenarios in comparison with scenarios in which unmanned ground vehicles operate, the impact of the accumulated positioning error on determination of the state variables' limits needs to be investigated in detail when the same action is repeated a couple of times. Furthermore, the inverse problem should also be considered, for a known pose tolerance and under imperfect sensing, as well as how to define adequate actions, and discretize the state space.

Acknowledgment

The results presented here are the research advancements supported by the Ministry of Science, Technological Development and Innovation of the Republic of Serbia under contract number 451-03-47/2023-01/ 200105 dated February 3, 2023.

References

[1] A. Singla, S. Padakandla, & S. Bhatnagar, "Memory-based deep reinforcement learning for obstacle avoidance in UAV with limited environment knowledge," *IEEE Transactions on Intelligent Transportation Systems*, vol. *22,* no. *1,* pp. *107-118,* November, 2019.

[2] Z. Yijing, Z. Zheng, Z. Xiaoyi, & L. Yang, " Q learning algorithm based UAV path learning and obstacle avoidence approach," *In 2017 36th Chinese control conference (CCC)*, Dalian, China, pp. *3397-3402,* July, 2017.

[3] D. Wang, et al., "A two-stage reinforcement learning approach for multi-UAV collision avoidance under imperfect sensing," *IEEE Robotics and Automation Letters*, vol. *5,* no. *1,* pp. *3098-3105,* April, 2022.

[4] Z. Miljković & Đ. Jevtić, "Object Detection and Reinforcement Learning Approach for Intelligent Control of UAV," in *Karabegović, I., Kovačević, A., Mandžuka, S. (eds) New Technologies, Development and Application V (NT 2022),* Switzerland: Springer, 2022, pp. *659-669*.

[5] R. S. Sutton & A. G. Barto, *Reinforcement Learning: An Introduction, 2nd* ed. Cambridge, Massachusetts: The MIT Press, 2018.

[6] M. Mitić, "Empirical control of an intelligent mobile robot based on machine learning," Ph.D. dissertation, Production Engineering Department, University of Belgrade - Faculty of Mechanical Engineering, Serbia, 2014.

[7] A. F. Sæther, B. Høye, K. D. Luong, O. B. Nygaard, "Autonomous Self-Learning Systems," College of Applied Science, Oslo and Akershus University, Norway, 2015.

 ISBN 978-86-7466-970-9